Computer Science Journal of Moldova, vol.**7**, no.3(21), 1999

PRONET: Basic concepts of a system of Artificial Intelligence

S. Lâsâi

Abstract

In the work are expounded the principles and basic elements of a system of artificial intelligence. Knowledge representation develops according to the method settled for processing. A thing, a phenomenon can be determined or established by more modules subject to their state as well as the links and relations between them. The system creates a set of blocks (modules) for which the concurrent work is pre-established. The volume of knowledge can be also increased without increasing the number of blocks.

For Artificial Intelligence systems the ratio between the user's and the system's work is too big.

The knowledge and data representation methods is the key of success of Artificial Intelligence. Traditional methods of knowledge representation are well known. Strong and week points of these methods are known too.

In the proposed system the representation and knowledge processing methods are elaborated at the same time and rely on the following concepts:

* determination of things. A thing or a phenomenon can be determined by more modules subject to their state as well as the links and relations between them;

* relations between names and things. A thing can be determined without knowing the name but under the relations and the content of cells of module. Other authors have also examined the relations between names and objects [1].

* the concurrent work of modules. The parallelism allows to elaborate either simpler or more efficient algorithms or both.

* self-learning. Some modules can analyze and perform changes within other modules.

* the property <<to forget>>. The system has the property <<to forget>> as the links are memorized in quasi-stacks for which is reserved a limited space. The system can be <<forced>> <<not to forget>>.

* the volume of knowledge. The application of the system in a domain requires a large number of modules. At the same time, the same set of modules allows to solve problems belonging to different classes. The volume of knowledge can be increased without increasing the number of blocks.

ProNet system is an application of L & R (Link and Relation) system [2] in Artificial Intelligence.

A version of L & R has been implemented in Turbo Prolog with a purpose to experiment with the work of basic elements of the system.

The ProNet system creates a set of blocks (modules) for a certain goal.

Each block contains:

* a set of cells (STATus, NAME, CLASS, ACTive, DEFine, CI (Cell Immediate), LAMP, TARG (TypeARG));

* an effector (procedure, demon);

* arguments;

* quasi-stacks 'in', 'out' and 'Bin'.

Cells, arguments and quasi-stacks are used to search, analyse and treat blocks.

The effector of the block contains operations which are executed when the effector is activated. Each block can contain one or more

ports. Ports are divided in sections. Each section contains 'condition' and 'operation' partitions (divisions). The purpose of dividing is the following:

* to facilitate the effector design
* to determine the ports and 'parasitic' sections (that don't work for a long period of time)
* the possibility to change blocks by other blocks at ports and sections level.

The concept of block is not new. The author has used it [3] as a method of languages realization.

The system can be either in 'learning' or 'activation' state. The blocks are defined in 'learning' state.

Arguments. Each argument contains four cells:

* argname. The cell contains the name of a block if <<argmod>> contains R. Otherwise we can write in the cell other values;
* argclass, that determines the class of <<argname >>;
* argdef, that determines whether <<argname>> is defined;
* argmod, that determines the type of argument.

The four types of blocks (F, V, L and T) are subject to types of arguments. Type F is a block with a fixed number of arguments. For type V the number of arguments is great and can be changed in activation state as well. The first argument is fictive. Type L (Like) differs from Type V (Variable) by activation method. When a block is defined by Type L the medium requests the value of p_limit. During the work the block self-activates when the number of defined arguments is greater or equal to p_limit. Type T (Table) can be compared with a relation of Relational Database. But blocks of type T contain also an effector and its attributes.

The system allows more methods to activate blocks.

Explicit method. Let the following operation be performed within the effector of block Bi:

set (act (name (Bt)), '3')

The expression means: <<set the value '3' for the block 'Bt' in cell ACT>>. In this case the medium writes (adds) the pair <Bi, 3> for Bt within 'Bin'. The medium will delete the less weighted pair if there is no space within 'Bin'. In the process of activation the medium checks if ports of the effector allows such activation. If activation is allowed then <Bi, W> is written for Bt within 'in', '1' is written within Lamp and block Bt starts to work. If Bi activates Bt with a higher frequency then W will be bigger and the pair <Bi, W> would have more chances to keep within 'in'. Blocks that are activated by Bt are registered in quasi-stack 'out'.

Implicit method. This method of activation is pre-established. The medium implements the method for blocks of type F. If a block B of class C passes to 'defined' state then the medium will try to activate all blocks of type F that contain an argument of class C.

Activation method pti. 'Virtual' blocks can be defined. The blocks contains one port. They can be sent into other blocks and activate them. 'Virtual' blocks execute as if being a party to these blocks. The notion pti (port immediate) differs from notions: macro, procedure or subroutine.

Logical expression and goal (<logical expression>) are used within an effector from 'condition'. Logical expressions are treated from intuitionism positions.

If goal (<logical expression>) is used in <<condition>> then the value of logical expression is calculated. If the value is false then operations from <<operation>> will be executed. The process repeats as much as the value of logical expression is false. Otherwise another section comes after. Blocks that contain goal (<logical expression>) are checked by the medium and their work can be interrupted. The effector can find out whether the execution of the block has been interrupted.

The congruencies $a = b$, $d = c$, $c = e$, $c = b$, $d = f$, $e = h$, $g = f$, $p = q$ are given. To verify the equality $b = h$. The problem has been solved in L and R using a stack. Other solution. Let's represent the

condition of the problem using a graph. We can easily find the solution.

We will proceed likewise in Pronet. We define $a, b, \ldots, q$ as blocks of class 'Tempo'. Such blocks can be added and eliminated in 'activation' state as well. Then we add arguments. The arguments of a block will be blocks linked to it. For example, the block c will contain arguments <b, d, e>.

The blocks <<start>> and <<final>> are used to solve some problems. The block <<start>> prepares data for the block that searches the solution. In our case the first argument is <b>. Then it activates the block that searches the solution. The block <<start>> activates block <<b>> using the operation:

set (pti (argname(arg (my))), name (seek_out))

where 'my' is the name of actual block (in this case – start) and the value of "argname(arg(my))" is 'b'.

The block 'Final' contains the goal (block) we search. The block 'Final' can activate other blocks.

The effector of block seek_arg contains sections as follows:

Port	Condition	Operation
	def (my) = 3	quit.% This block was treated (processed)
	argdef (arg (name (final))) = 3	quit.% The solution was found (anteriorly)
	argname (arg(name (final)))=my	set(def(my),3), set (arg def(arg(name(final))),3) quit.% Now we have found solution
		blank (p), set (ci (my), argname (p (arg (my)))).
	goal (not (e_name (ci (my))))	set (pti (ci (my)), name (seek_arg)), new (p), set (ci (my), argname (p (arg (my)))).
		Init (p), set (def (my), 3), quit.

This block contains one port without input conditions and six sections. Let's start from section 4. The operation blank (p) - neutralizes the value of p. New(p) sets a new value for p so it will point to the previous argument. The cell <<ci>> is used for a compact writing. The block <<c>> contains 3 arguments: b,d, and e. In this case ci (my) is equivalent to ci (name(c)). First we put 'b' within ci (my). Then 'd' and 'e'. We check if the condition contains block 'b'. The operation "set (pti (ci (my)), name (seek_arg))" sends the block 'seek_org' to 'b', 'd' and 'e' so it sends the block. If the solution is found then '3' will be sent for 'Final' to argdef. The block 'Final' usually ends the work with:

set (act (in (name (start))), 3), quit

where "in (name (start))" is the name of the block that activates the block 'start'.

If we use more processors then 'Final' must contain a section with the condition:

goal (not (e_name (class (Tempo), lamp (1))))

The expression will be true if none of blocks of class 'Tempo' works. This is an indispensable condition for case a=q. If we use one processor then it makes no sense to use lamp (-). In this case we can use the operation "close (class (Tempo))" and so all blocks of class 'Tempo' will be processed before 'Final'.

It is not reasonable to prove a theorem every time we use it. Let blocks A, B and C be linked.

Block B has one 'entance' and one 'exit'. If C is the application of a theorem and B- its proof then from beggining we can include the following section into the effector of bloc B:

Condition	Operation
def (my) = 3, e_name (out (my)).	Set (act (out (my)),3).

Then after the first good try of B (B will pass to 'defined' state) the proof will not be repeated. So we pass directly to C. Another effector can localize structures of type $< A, B, C >$. The effector can inform us which blocks pretend to theorem status. If necessary, the effector can change the structure.

References

[1] Sapiro E., Takenchi A., *Object oriented programming in Concurrent Prolog*, New Generation Computing, 1 (1983) OHMSHA, LTD, pp.25-48;

[2] Lâsâi S., *L and R – un system formel pour les deductions*, Computer Science. The proceedings of the 3rd International Symposium of Economic Informatics – May 1997, Bucharest. pp.65-69.

[3] Lâsâi S., *G1 – self-extensible geometric language and its implementation*, Computer software systems, Kiev: Naukova Dumka, 1973, pp.51-63. (in Russian)

S.Lâsâi,
Moldova State University,
Faculty of Mathematics and Computer Science,
60, Mateevich str., Kishinev,
MD-2009, Moldova

Received November 22, 1999